View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio de Objetos de Docencia e Investigación de la Universidad de Cádiz

ESCUELA SUPERIOR DE INGENIERIA

Programación en Internet

Grado en Ingeniería Informática

Tutorial sobre como añadir documentación Swagger a un servicio REST

Autor:
Jose Antonio Caravaca Diosdado

Supervisor:
Guadalupe Ortiz Bellot

Cádiz, 05 de Octubre de 2016

Índice

1. Introducción

En este tutorial aprenderemos a añadir documentación Swagger a un servicio REST implementado con java, se entiende que el usuario sabe como crear el servicio, por lo que, aunque se mostrará el servicio que usaremos como ejemplo, no se explicará como crearlo, solo los pasos necesarios para añadir la documentación Swagger.

2. Nuestro servicio

A continuación mostraremos el servicio al cual añadiremos la documentación Swagger. Mostraremos únicamente el archivo web.xml y las clases java que necesitamos.

2.1. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <servlet>
    <servlet-name>Mi Servicio Hello REST</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>nombrePaquete,com.fasterxml.jackson.jaxrs.json</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>Mi Servicio Hello REST</servlet-name>
    <url-pattern>/demo/*</url-pattern>
  </servlet-mapping>
</web-app>
```

2.2. Hello.java

En esta clase definimos los métodos ofrecidos por el servicio REST

```
package nombrePaquete;

import java.util.HashMap;
import java.util.Map;

import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class Hello{
@GET
@Produces(MediaType.TEXT_PLAIN)
public String sayPlainTextHello(){return "Hola Mundo";}

@GET
@Path("/hello2")
@Produces(MediaType.TEXT_PLAIN)
public String sayPlainTextHello2(){return "Hello Plain 2";}

@GET
@Path("/helloId/{oid}")
@Produces(MediaType.TEXT_PLAIN)
public String sayHelloWithId(@PathParam("oid")int id)
{return "Hola " + id;}

@GET
@Path("/dateJSON")
@Produces({"application/json"})
public MyDate getDate_JSON() {
MyDate oneDate = new MyDate();
oneDate.setDay(25);
oneDate.setMonth(12);
oneDate.setYear (2014);
return oneDate;}

@POST
@Path("/name")
@Consumes(MediaType.TEXT_PLAIN)
@Produces(MediaType.TEXT_PLAIN)
public String HelloName(String myName){
return "Hello "+myName;}

@POST
@Path("/myDate2015")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public MyDate dateToString(MyDate myDate){
myDate.setYear(2015);
```

```
return myDate;}

private static Map<String, MyDate> myMap = new HashMap<>();
static{
MyDate myDate = new MyDate();
myDate.setDay(25);
myDate.setMonth(12);
myDate.setYear(2014);
myMap.put("Navidad", myDate);
myMap.put("Nochebuena", myDate);
myMap.put("AnioNuevo", myDate);
myMap.put("Reyes", myDate);

  }

@GET
@Path("/importantDates")
@Produces({"application/json"})
  public Map<String, MyDate> getImportantDates()
{
return myMap;}

@GET
@Path("/getDate/{date}")
@Produces({"application/json"})
  public MyDate getDate(@PathParam("date") String key)
{
MyDate date =myMap.get(key);
return date;}

@DELETE
@Path("/deleteDate/{date}")
@Produces({MediaType.TEXT_PLAIN})
  public String deleteDate(@PathParam("date") String key)
{
myMap.remove(key);
return "Done";}

@POST
@Path("/addDate/{date}")
@Consumes({"application/json"})
  public void addImportantDate(@PathParam("date") String key, MyDate myDate)
{
myMap.put(key, myDate);}

@PUT
@Path("/modifyDate/{date}")
@Consumes({"application/json"})
 public void modifyImportantDate(@PathParam("date") String key, MyDate myDate)
{
myMap.put(key, myDate);
}

}
```

2.3. MyDate.java

Clase auxiliar para el manejo de datos tipo JSON

```
package nombrePaquete;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class MyDate {

private int day;
private int month;
private int year;

public int getDay() {return day;}
public void setDay(int day) {this.day = day;}
public int getMonth() {return month;}
public void setMonth(int month) {this.month = month;}
public int getYear() {return year;}
public void setYear(int year) {this.year = year;}
}
```

3. Librerías

Para que nuestra documentación funcione necesitamos tener actualizada la librería jackson, en nuestro caso usaremos la versión 2.8, además debemos incluir los siguientes archivos jar en nuestro proyecto:

- common-lang3.jar
- slf4j-api-1.7.21.jar
- jersey-media-multipart-2.23.2.jar
- swagger-annotations-1.15.10.jar
- swagger-core-1.5.0.jar
- swagger-jaxrs-1.5.0.jar
- swagger-jersey2-jaxrs-1.5.0.jar
- swagger-models-1.5.0.jar

4. Configuración

4.1. web.xml

En nuestro archivo web.xml debemos añadir las siguientes configuraciones:
En el parámetro jersey.config.server.provider.packages debemos añadir el paquete io.swagger.jaxrs.listing, de forma que la configuración quedaría de este modo

```
<init-param>
  <param-name>jersey.config.server.provider.packages</param-name>
  <param-value>
  nombrePaquete,com.fasterxml.jackson.jaxrs.json,io.swagger.jaxrs.listing</param-value>
</init-param>
```

Añadimos un servlet en el que indicaremos el nombre de nuestra clase principal con la siguiente configuración:

```
<servlet>
  <servlet-name>jersey</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>jersey.config.server.provider.classnames</param-name>
    <param-value>
      io.swagger.jaxrs.listing.ApiListingResource,
      io.swagger.jaxrs.listing.SwaggerSerializers,
      nombrePaquete.Hello
    </param-value>
  </init-param>
</servlet>
```

Otro servlet indicando el path y la versión de nuestra Api, recuerda sustituir el path que se muestra a continuación (http://localhost:8080/HelloWorldSwagger) por el de tu servicio:

```
<servlet>
  <servlet-name>Jersey2Config</servlet-name>
  <servlet-class>io.swagger.jersey.config.JerseyJaxrsConfig</servlet-class>
  <init-param>
    <param-name>api.version</param-name>
    <param-value>1.0.0</param-value>
  </init-param>
  <init-param>
    <param-name>swagger.api.basepath</param-name>
    <param-value>http://localhost:8080/HelloWorldSwagger</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>
```

Y por último, un filter que se encargará de habilitar los permisos CORS para que podamos visualizar nuestra Api en el cliente de swagger:

```
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Una vez configurado el archivo web.xml, si ponemos en marcha nuestro servicio y accedemos desde nuestro navegador a `http://localhost:8080/HelloWorldSwagger/demo/swagger.json`, se nos mostrará algo parecido a esto:

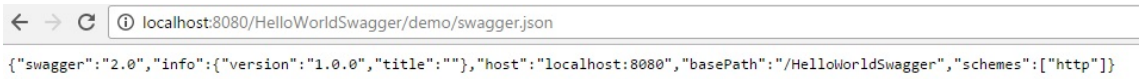

Figura 1: Json con la configuración de Swagger

4.2. Hello.java

Ya podemos comenzar a documentar nuestro servicio, como ejemplo en este tutorial usaremos las anotaciones @Api, @ApiOperation y @ApiParam, pero podeis usar muchas más para que vuestra Api esté lo mejor documentada posible, en este enlace podeis consultar todas las anotaciones que incluye Swagger: `https://github.com/swagger-api/swagger-core/wiki/Annotations`

Primero importamos las librerías necesarias:

```
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
```

Y luego incluimos las anotaciones que queramos en nuestro código:

```
@Path("/hello")
@Api(value = "/hello")
public class Hello{
        .
        .
        .
@GET
@Path("/hello2")
@Produces(MediaType.TEXT_PLAIN)
@ApiOperation(
        value = "Saluda",
        notes = "Dice Hello Plain 2"
)
public String sayPlainTextHello2(){return "Hello Plain 2";}
        .
        .
        .
@GET
@Path("/helloId/{oid}")
@Produces(MediaType.TEXT_PLAIN)
@ApiOperation(
        value = "Saluda",
        notes = "Dide Hola + id"
)
public String sayHelloWithId(
   @ApiParam(value = "ID del usuario", allowableValues = "range[1," +
   Integer.MAX_VALUE + "]", required = true)
   @PathParam("oid")int id)
{return "Hola " + id;}
```

¡OJO!: La anotación @Api es siempre obligatoria, y la anotación @ApiOperation es indispensable para probar tus servicios en el cliente de Swagger

Una vez hayamos finalizado nuestra documentación, si arrancamos nuestro servidor y accedemos a `http://localhost:8080/HelloWorldSwagger/demo/swagger.json` esta vez se nos mostrará un poco más de información:

Figura 2: Json con la configuración de nuestro servicio

Pero si alguien quiere consultar la documentación de tu servicio y ve esto, propablemente no se entere de nada, así que vamos a mostrarlo un poco más bonito. Nos dirigiremos al cliente de Swagger: `http://petstore.swagger.io/` ,introduciremos nuestro path en su buscador y hacemos click en el nombre de nuestra Api. Ahora si, ya tenemos nuestra documentación con un diseño agradable, fácil de entender y con posibilidad de probar nuestros servicios desde este mismo cliente.

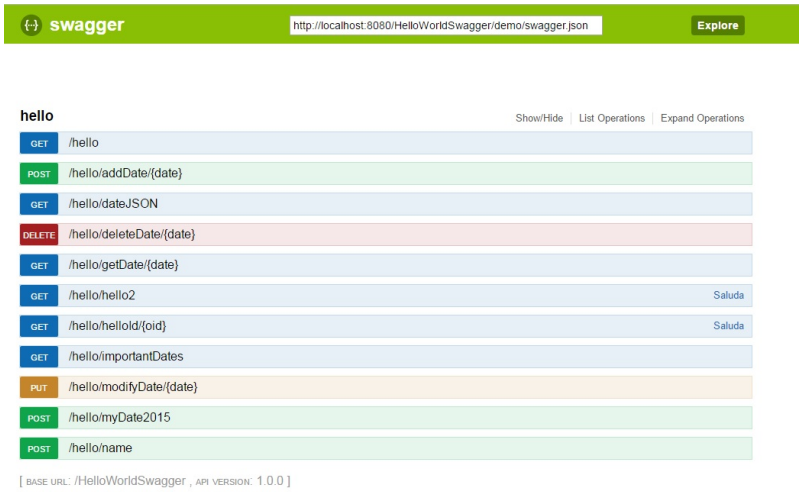

Figura 3: Cliente Swagger con los métodos de nuestro servicio

Figura 4: Detalles de un servicio en el cliente Swagger